

# DBTLAB – Introduction

Rudi Poepfel Lemaitre

based on material from Volker Markl, Viktor Rosenfeld, Jonas Traub, Martin Kiefer

# Rudi Poepsel Lemaitre

## **BSc Computer Science**

TU Berlin (2015-2018)

## **MSc Computer Science**

TU Berlin (2018-2021)

## **Software Campus Project**

Holtzbrinck + TU Berlin (2020-2021)

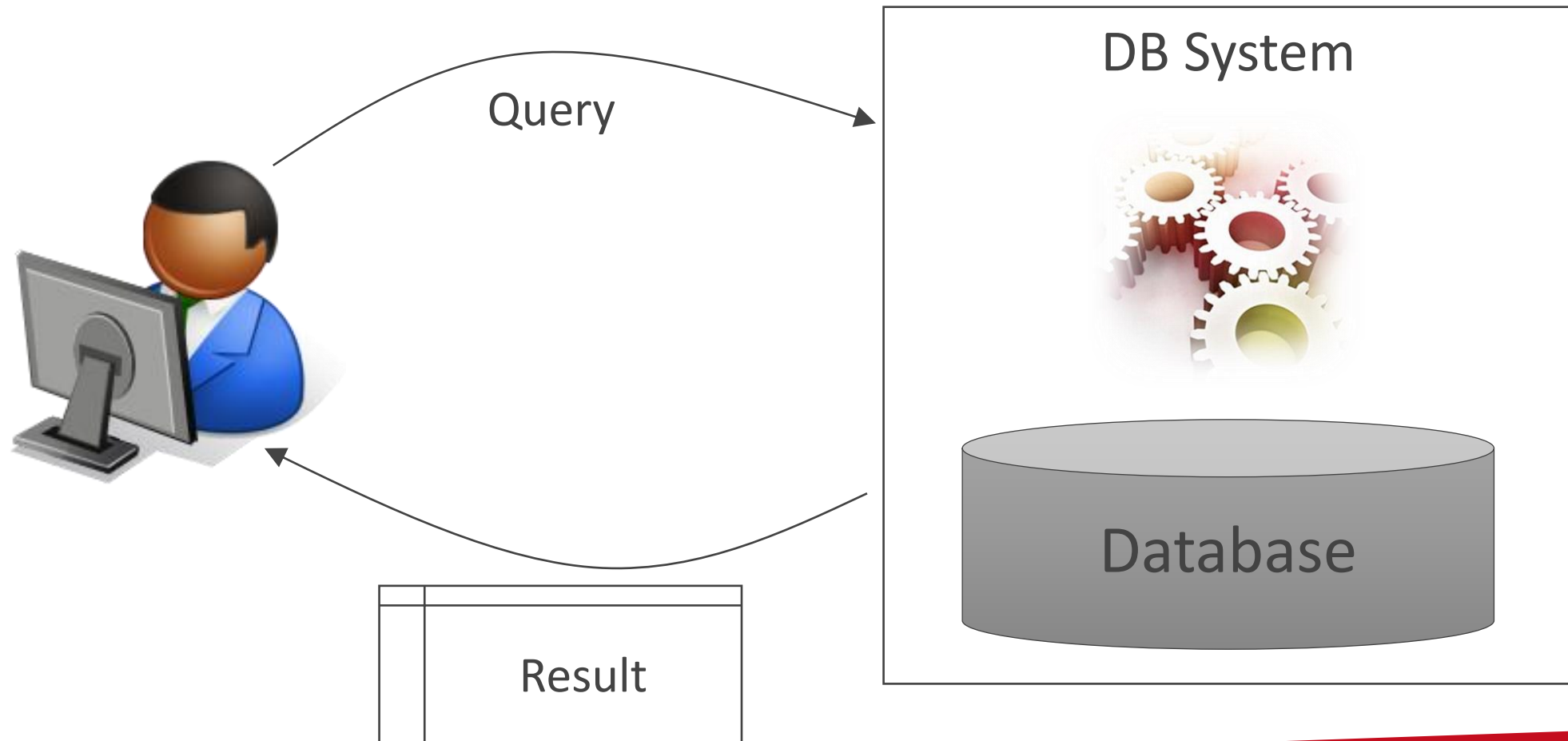
## **Ph.D. Computer Science**

TU Berlin (Since July 2021)



# Motivation

# Databases from the perspective of the user



## How to process the query?

```
select
  nation, o_year, sum(amount) as sum_profit
from (
  select
    n_name as nation,
    extract(YEAR FROM o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) -
      ps_supplycost * l_quantity as amount
  from
    part, supplier,
    lineitem, partsupp,
    orders, nation
  where
    s_suppkey = l_suppkey
  and ps_suppkey = l_suppkey
  and ps_partkey = l_partkey
  and p_partkey = l_partkey
  and o_orderkey = l_orderkey
  and s_nationkey = n_nationkey
  and p_name >= 'g' and p_name < 'h'
) as profit
group by nation, o_year
order by nation, o_year desc
```



|           |      |               |
|-----------|------|---------------|
| ALGERIA   | 1998 | 29030622.6876 |
| ALGERIA   | 1997 | 52008575.1366 |
| ALGERIA   | 1996 | 51877580.0470 |
| ALGERIA   | 1995 | 49822895.3401 |
| ALGERIA   | 1994 | 50869151.1316 |
| ALGERIA   | 1993 | 51194080.5894 |
| ALGERIA   | 1992 | 52411129.6916 |
| ARGENTINA | 1998 | 27491032.3173 |
| ARGENTINA | 1997 | 45934161.0309 |
| ARGENTINA | 1996 | 45884131.9124 |

# How to process the query?

```
select
  nation, o_year, sum(amount) as sum_profit
from (
  select
    n_name as nation,
    extract(YEAR FROM o_orderdate) as o_year,
    l_extendedprice * (1 - l_discount) -
      ps_supplycost * l_quantity as amount
  from
    part, supplier,
    lineitem, partsupp,
    orders, nation
  where
    s_suppkey = l_suppkey
    and ps_suppkey = l_suppkey
    and ps_partkey = l_partkey
    and p_partkey = l_partkey
    and o_orderkey = l_orderkey
    and s_nationkey = n_nationkey
    and p_name >= 'g' and p_brand < 'a'
  ) as profit
group by nation, o_year
order by nation, o_year desc
```

How to parse  
the query?

How is the data  
stored?

Which index to  
use?

What data to  
cache?



|           |      |               |
|-----------|------|---------------|
| ALGERIA   | 1998 | 29030622.6876 |
| ALGERIA   | 1997 | 52008575.1366 |
| ALGERIA   | 1996 | 51877580.0470 |
| ALGERIA   | 1995 | 49822895.3401 |
| ALGERIA   | 1994 | 50869151.1316 |
| ALGERIA   | 1993 | 51194080.5894 |
| ALGERIA   | 1992 | 52411129.6916 |
| ARGENTINA | 1998 | 27491032.3173 |
| ARGENTINA | 1997 | 45934161.0309 |
| ARGENTINA | 1996 | 45884131.9124 |

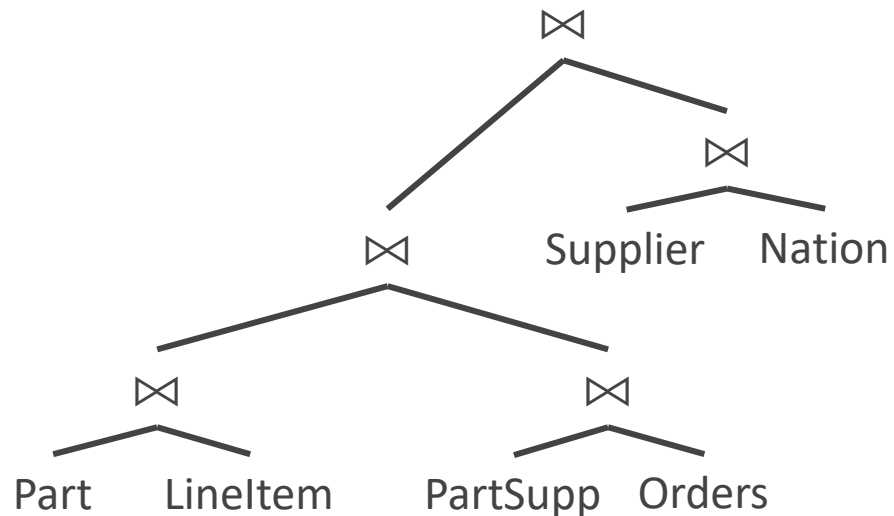
Join the tables  
in which order?

Use which join  
algorithm?

When to sort  
the data?

How to  
aggregate data?

## How many different join orders exists for n tables?



- Join order can be expressed as binary tree.
- Each possible tree is a join order.
- # binary trees with n leaves: Catalan number  $C_{n-1}$
- $C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
- $C_{n-1}$  possible trees for each possible permutation of tables
- $n!$  possible permutation of tables
- **Total number of join orders** =  $n! \frac{(2(n-1))!}{(n-1)!n!} = \frac{(2n-2)!}{(n-1)!}$

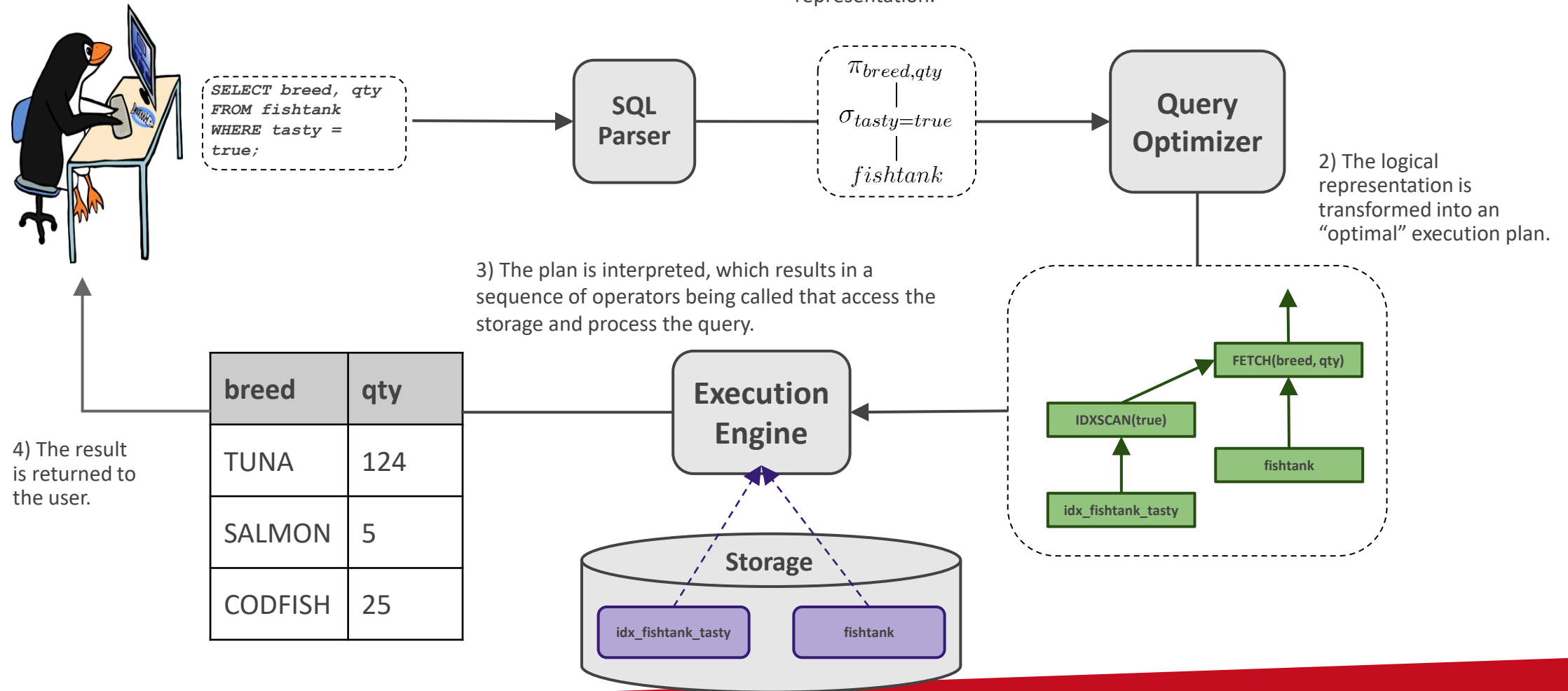
## How many different join orders exists for n tables?

| Number of tables | Possible join orders |
|------------------|----------------------|
| 1                | 1                    |
| 2                | 2                    |
| 3                | 12                   |
| 4                | 120                  |
| 5                | 1,680                |
| 6                | 30,240               |
| 7                | 665,280              |
| 8                | 17,297,280           |
| 9                | 518,918,400          |
| 10               | 17,643,225,600       |

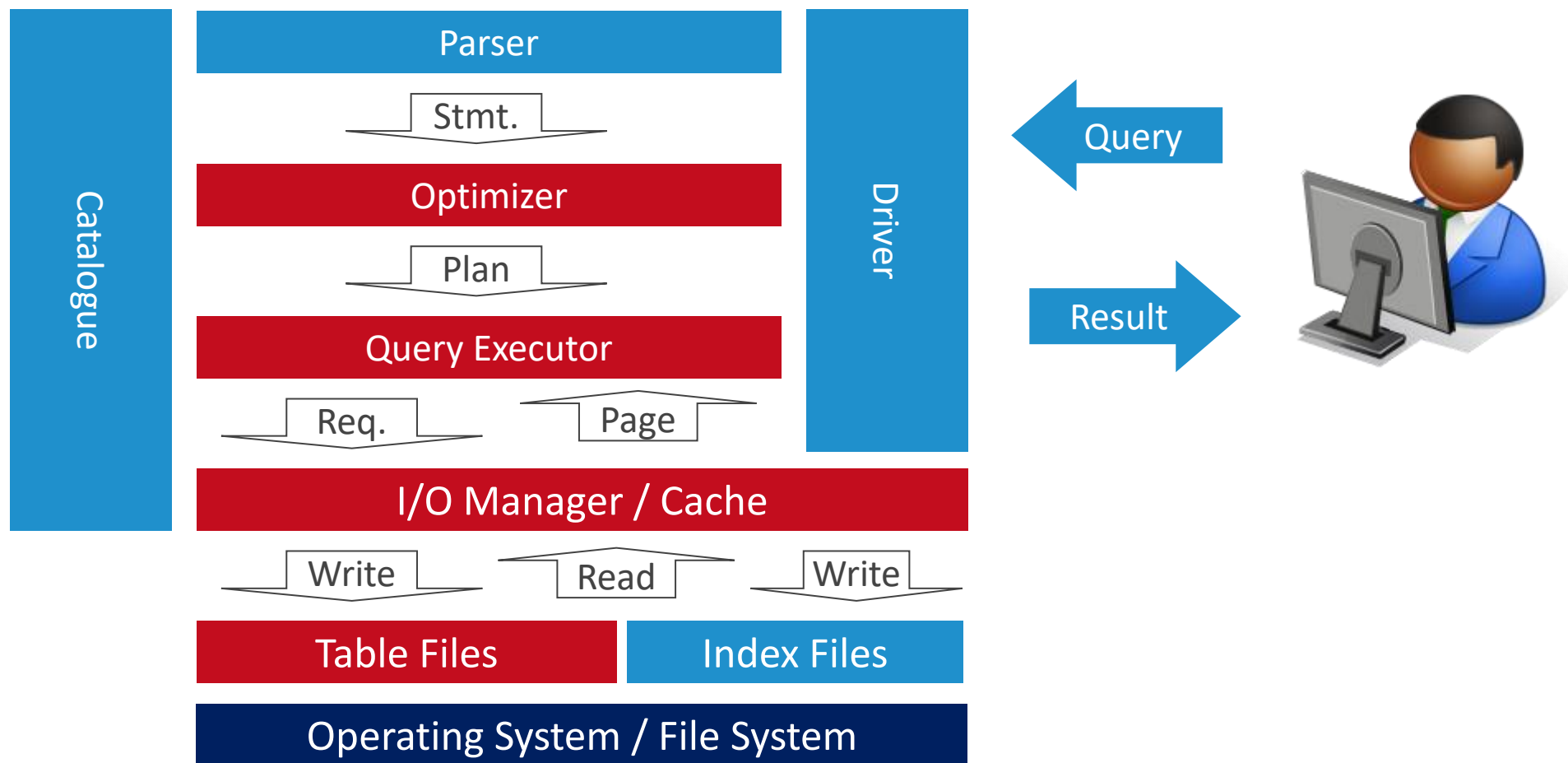
- Can't just pick any join order
- **Orders of magnitude performance difference**



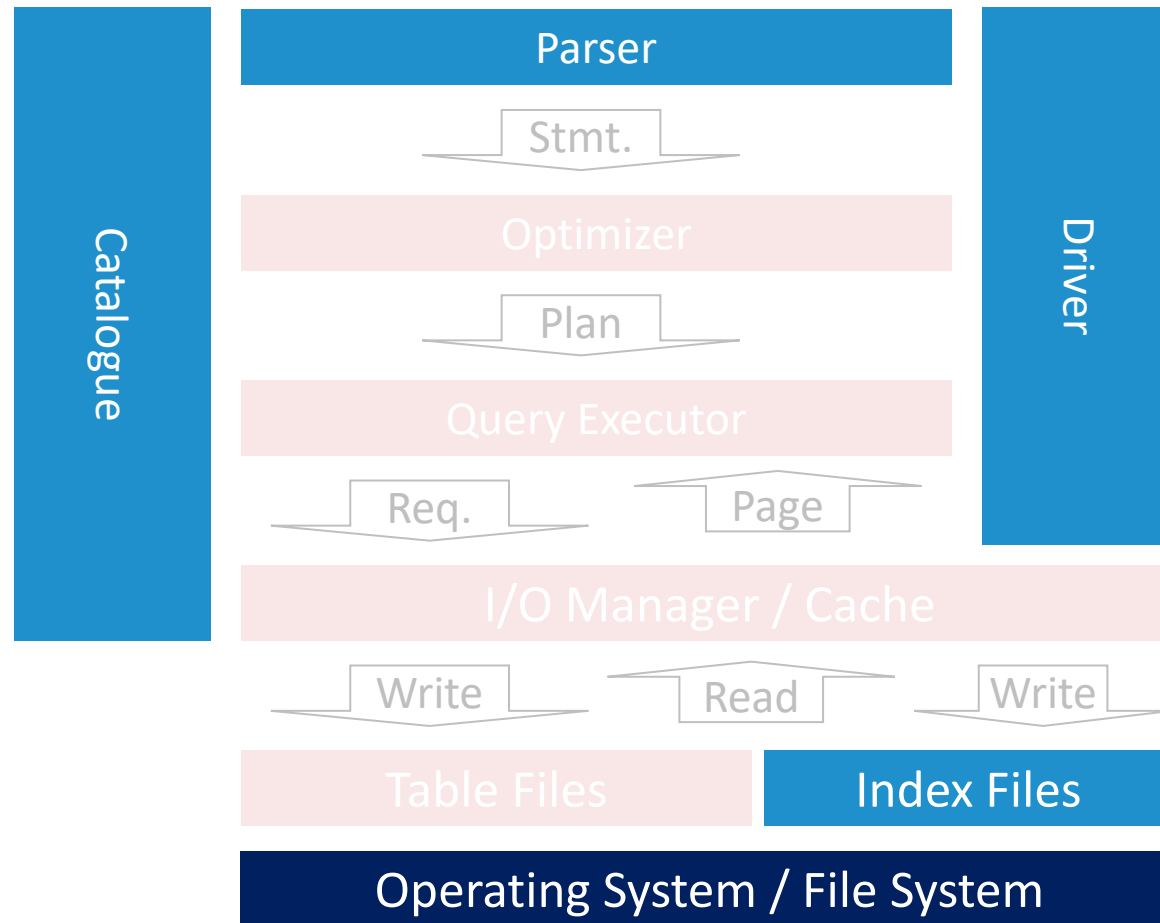
# Query execution – The big picture



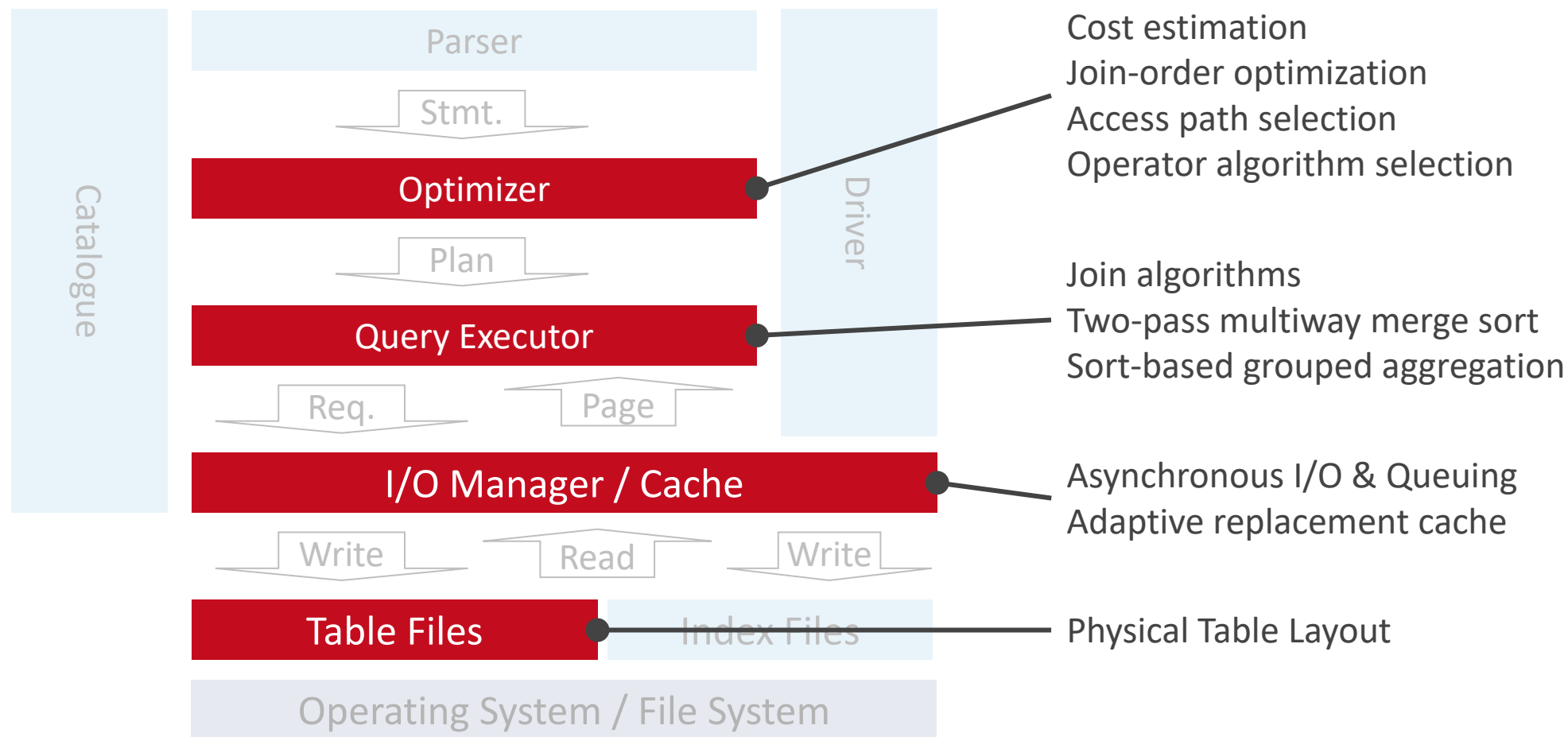
# Basic system architecture



We will give you ...



# You will implement ...



# A complete database management system

- Read-Dedicated System
  - Multiple Concurrent Read Queries
  - Single Updating (Roll In) Query – Snapshot Isolated
- Supports simple SQL
  - SELECT – FROM – (WHERE – GROUP BY – HAVING – ORDER BY)
  - INSERT, DELETE, UPDATE
- Aggregations: SUM, COUNT, MIN, MAX, AVG
- Nested Table Expressions (Simple Subqueries)
- Data Types: SMALLINT, INT , BIGINT, FLOAT, DOUBLE, CHAR, VARCHAR, DATE, TIME, TIMESTAMP
- Predicates
  - Conjunctive Normal Form
  - Operators: [=, <, >, <=, >=, <>]

# Course Organization

## General

- This is a programming course.
  - You are given interfaces and support code.
  - You hand in your implementation for automatic evaluation.
- Each exercise is accompanied by a bit of theory.
  - Not DBT – Just what you need for our implementation.
- You work in groups of three students.
- Support through the ISIS forum and during the weekly lecture on Thursday

# Groups

- You will work in **groups of exactly 3 students!**
- Form groups by the next lecture and pick a group on the evaluation server.
- Use today or the forum to find groups.
- Why groups? Because groups are awesome!
- You have to write *and debug* a lot of code.
- If you get stuck, a group member can pick up the work.
- If your peer quits and leaves you alone, team up with other students in the same situation.





## This course is **hard**

- You will have to write non-trivial code
- You will have to understand unit tests
- You will get stuck
- 6 ECTS = 180 hours = **12 hours per week per student**
- Take this course if you ...
  - Have worked with code from other people before
  - Feel confident in programming Java (including multi-threading)
  - Can tolerate frustrating debugging sessions
- You will ...
  - Learn the internals of a database system
  - Get valuable experience working on existing projects

Thank you for your attention and see you next week!